ACADEMIC PRESS

Available at www.ComputerScienceWeb.com
POWERED BY SCIENCE DIRECT®

Computer Vision and Image Understanding 90 (2003) 258–294

Computer Vision and Image Understanding

www.elsevier.com/locate/cviu

Implementation techniques for geometric branch-and-bound matching methods

Thomas M. Breuel

Xerox Palo Alto Research Center, 3333 Coyote Hill Road, Palo Alto, CA 94304, USA

Received 24 September 2001; accepted 6 February 2003

Abstract

Algorithms for geometric matching and feature extraction that work by recursively subdividing transformation space and bounding the quality of match have been proposed in a number of different contexts and become increasingly popular over the last few years. This paper describes matchlist-based branch-and-bound techniques and presents a number of new applications of branch-and-bound methods, among them, a method for globally optimal partial line segment matching under bounded or Gaussian error, point matching under a Gaussian error model with subpixel accuracy and precise orientation models, and a simple and robust technique for finding multiple distinct object instances. It also contains extensive reference information for the implementation of such matching methods under a wide variety of error bounds and transformations. In addition, the paper contains a number of benchmarks and evaluations that provide new information about the runtime behavior of branch-and-bound matching algorithms in general, and that help choose among different implementation strategies, such as the use of point location data structures and space/time tradeoffs involving depth-first search.
© 2003 Elsevier Science (USA). All rights reserved.

Keywords: Geometric matching; Maximum likelihood; Gaussian error; Bounded error; Branch and bound; Global optimization; Visual object recognition

1. Introduction

Matching geometric primitives under a variety of transformations is an important problem in computer vision, robotics, and many other applications. Over the last

E-mail address: tbreuel@parc.xerox.com.

1077-3142/03/$ - see front matter © 2003 Elsevier Science (USA). All rights reserved.
doi:10.1016/S1077-3142(03)00026-2

decade, a number of techniques based on branch-and-bound methods have been proposed for finding globally optimal solutions to such matching problems. Such techniques work by recursively subdividing the space of transformation parameters and computing upper bounds on the possible quality of match within each subregion, and they can guarantee returning globally optimal solutions to geometric matching problems. While a number of individual applications of, and approaches to, branch-and-bound methods to geometric matching have been published in the literature [6,18,23,26,29], this paper attempts to review and bring together information useful for the application of branch-and-bound methods to a wide variety of geometric matching problems under different error measures and transformations in a single place and framework.

While the reference material and experimental results have applications to other branch-and-bound techniques, the paper focuses on the use of *matchlists* in the evaluation and bounding of quality of match functions. Matchlists simplify the implementation of branch-and-bound algorithms for geometric matching, obviating the need for point location data structures or discrete distance transforms. This enables the practical solution of previously difficult or unsolved problems. Among others, the paper describes the following new techniques:

- A method for globally optimal partial line segment matching under bounded error or Gaussian error models. The previously best method for globally optimal line segment matching is probably the method described by Yi and Camps [31], but that method has no provisions for partial line segment matches and requires the computation of a four-dimensional distance transform. The method presented in this paper allows for partial matches and requires no separate distance transform computation. It is one example of transform-dependent and non-circular error bounds, for which no globally optimal algorithms appear to have been previously described. Experiments on the LiME dataset [3] are presented.
- An approach to globally optimal matching under a Gaussian error model using matchlists. Unlike the method described by Jurie [23], the approach requires no separate local search, and unlike the technique proposed by Olson [29], it yields subpixel accurate results without the need for an approximate interpolative step or a distance transform. Furthermore, the method improves on previous approaches [18,26,29] in that it incorporates, and prunes the search using, a precise error model of feature orientations.
- A simple and robust technique for finding multiple distinct object instances is described, related to the method described in [11].

Several additional uses enabled through matchlists are mentioned throughout the paper.

In addition to these results, the paper also contains a number of experimental evaluations and comparisons:

- A direct comparison of the performance of matchlist-based and point location-based techniques, demonstrating that matchlist-based techniques are not only simpler, but also faster.
- An empirical comparison of depth-first and best-first search strategies in these geometric matching algorithms. Substantially lower memory requirements at only

modestly increased running times are demonstrated. Demonstrating this tradeoff allows the depth-first approach to be used on memory-constrained platforms like handhelds and embedded devices.
- An empirical comparison of efficient implementation of alignment and branch-and-bound methods. These results have implications for attempts to speed up branch-and-bound methods using alignment methods, discussed in more detail below.
- A more detailed analysis of the runtime behavior of branch-and-bound methods than that previously presented by Mount et al. [26].

These experimental evaluations provide new information about the runtime behavior of branch-and-bound matching algorithms in general, information about the specific tradeoffs between matchlist and non-matchlist approaches, and they illuminate a number of other important tradeoffs and implementation choices available to users of such methods.

The overall goal of the paper is to provide the reader with useful guidelines, baseline performance data, and reference information for implementing branch-and-bound geometric matching algorithms in a wide variety of existing and novel applications.

2. An instance of geometric matching

For simplicity of exposition, let us begin with a specific instance of the recognition problem: bounded error matching of planar point sets under isometric transformations. We will generalize this model to other kinds of error measures and transformations in Sections 9 and 10.

Define a model to be a set of points $M = \{m_1, \ldots, m_r\} \subseteq \mathbb{R}^2$ and an image to be another set of points $B = \{b_1, \ldots, b_s\} \subseteq \mathbb{R}^2$. We consider a bounded set of isometric transformations $\mathbb{T}_{\text{all}}$; that is, the set of transformations $T : \mathbb{R}^2 \to \mathbb{R}^2$ consisting of translations and rotations and parameterize these transformations by a vector $(\Delta x, \Delta y, \alpha)^{\mathrm{T}} \in [\Delta_{\min}, \Delta_{\max}] \times [\Delta_{\min}, \Delta_{\max}] \times [0, 2\pi) \subseteq \mathbb{R}^3$. We also assume a bounded error notion of quality of match $Q(T; M, B, \epsilon)$ under the Euclidean distance; where it is clear from context, we will omit the dependence of Q on M, B, and/or ϵ. That is, the quality of match assigned to a transformation T is the number of model points the transformation brings within an error bound of ϵ of some image point. If we write $\lfloor \textit{predicate} \rfloor$ for the standard indicator function, which assumes the value 1 if the *predicate* is true, 0 otherwise, we can write this quality of match function as

$$Q(T) = \sum_{m \in M} \max_{b \in B} \lfloor \|T(m) - b\| < \epsilon \rfloor. \tag{1}$$

This may seem like an unnecessarily complex way of expressing a count of the number of matching features, but it is a form that easily generalizes to other error measures. One way of defining optimal geometric matching is to find a transformation $T_{\max}$ (usually not unique) that maximizes the quality of match for a given M, B, and ϵ:

$$T_{\max}(M, B, \epsilon) = \arg \max_{T \in \mathbb{T}_{\text{all}}} Q(T; M, B, \epsilon). \tag{2}$$

We will generalize this model in subsequent sections to a large variety of different spaces of transformations and error models.

3. A random model of problem instances

Before proceeding to a description of algorithms for solving the bounded error matching problem, let us look at a particular distribution of random problem instances of the geometric matching problem.

Let each model consist of 20 points uniformly sampled from the rectangle $[-100, 100] \times [-100, 100]$. Each image consists of 10 points randomly selected from the model, rotated by a random angle in the interval $[0, 2\pi)$ and translated by a random translation drawn from $[0, 512] \times [0, 512]$. Each such model point is additionally perturbed by an error vector randomly and uniformly selected from the set $\{v \in \mathbb{R}^2 : \|v\| < 5\}$; this represents a 5% error on the location of model features in the image. Additionally, each image contains a random number (between 10 and 160 in different experiments) of background points, uniformly selected from the rectangle $[0, 512] \times [0, 512]$. The image points (both model and background points) are randomly permuted before being used as input to the matching algorithms. Fig. 1 shows an example of such a random problem instance.

These kinds of random problem instances are useful for several reasons. First, they approximate[1] the distribution of problem instances for which optimal bounded error matching returns maximum likelihood solutions [22]. (If we perturb model points with error vectors with a Gaussian distribution, then the maximum likelihood solution to the matching problem corresponds to robust least square matching.)

Note that there is no orientation information associated with any of the points in this model. In natural images, orientation information associated with feature points can often be derived from local gradients or geometric primitives. Given the accuracy with which such information can be extracted, such orientation information greatly cuts the number of possible correspondences between image and model points during the search and results in significant speedups. The reason for not using feature orientations is that, while feature orientations are easy to incorporate precisely into the matchlist-based approach described below, their use in other branch-and-bound methods is more complex and often heuristic. The use of feature orientations would therefore complicate the interpretation of benchmark results and generally disadvantage non-matchlist-based approaches.

From a practical point of view, samples drawn from this distribution of problem instances are simple to generate and reproduce and have been used by other researchers for testing geometric matching algorithms (e.g. [26]). A number of numerical experiments and benchmarks in the remainder of this paper are based on this test case.

[1] The correspondence is only approximate because of boundary effects.

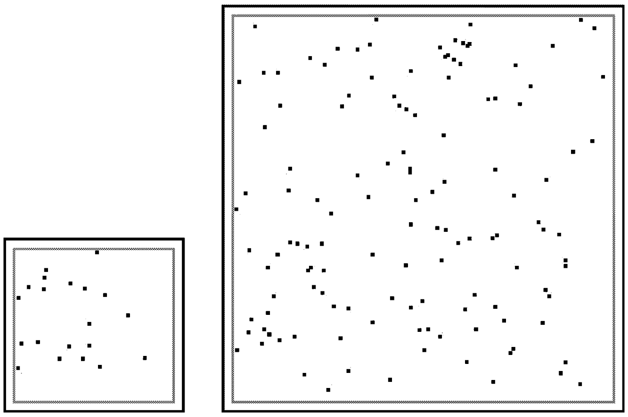

Fig. 1. A random instance of geometric matching problems used for various empirical performance measurements throughout the paper. The model is described in Section 3.

4. A branch-and-bound approach

We can organize the search for a globally optimal solution to Eq. (2) as follows.

Algorithm 1.

1. The algorithm maintains a priority queue of search states. When two search states have the same priority, the state with the lower depth in the search tree is preferred. The queue is initialized with a state representing all possible solutions.
2. Each search state is associated with a subregion of transformation space $\mathbb{T}_k \subseteq \mathbb{T}_{\text{all}}$. It is further associated with an upper bound $\forall T \in \mathbb{T}_k : \hat{Q}_k = \hat{Q}(\mathbb{T}_k) \geqslant Q(T)$; the upper bound serves as the priority of the state. For termination and correctness, the upper bound needs to satisfy at least the condition that $\hat{Q}(\{T\}) = Q(T)$.
3. The algorithm removes the state with the highest upper bound from the priority queue. In case of ties, states with lower depth in the search tree are preferred.
4. Pick some transformation $T \in \mathbb{T}_k$; if $Q(T) = \hat{Q}(\mathbb{T}_k)$, terminate the search and return T as a solution.
5. Otherwise, we split the region $\mathbb{T}_k$ into two disjoint subregions $\mathbb{T}_{2k}$ and $\mathbb{T}_{2k+1}$ such that $\mathbb{T}_k = \mathbb{T}_{2k} \cup \mathbb{T}_{2k+1}$ along its largest dimension and insert these subregions back into the priority queue.

This type of algorithm is known as a *branch-and-bound* algorithm.

It should be noted that the algorithm makes no reference to a pixel grid, either in its image or model points, or in any point location data structure. Rather, coordinates are given as floating point numbers, and, within the limits of floating point precision and search termination/tolerance criteria, solutions returned by the algorithm are subpixel accurate. (Note also that in the case of bounded error matching, the range of the quality of match function is discrete and consists of the non-negative integers, a fact that we take advantage of later.)

The three questions that we might immediately ask about this kind of algorithm are whether the solutions it returns are optimal, whether it always terminates, and what its complexity is. Let us sketch the relevant arguments here for the case of globally optimal bounded error matching using a branch-and-bound approach.

Theorem 1. *When Algorithm* 1 *terminates, it returns a globally optimal solution.*

Initially, the algorithm considers the set of all possible transformations $\mathbb{T}_{\text{all}}$, which, by definition of being globally optimal over $\mathbb{T}_{\text{all}}$, contains at least one globally optimal transformation $T_{\max}$. Therefore, at the beginning of the algorithm, the priority queue contains a region containing an optimal solution.

We need to establish first that this property is a loop invariant, i.e., that the globally optimal solution does not simply disappear from the priority queue. When we remove a region $\mathbb{T}_k$ and its associated upper bound $\hat{Q}(\mathbb{T}_k)$ from the priority queue, either $\mathbb{T}_k$ contains $T_{\max}$ or it does not. If it contains the globally optimal solution $T_{\max}$ and we subdivide $\mathbb{T}_k$ into two disjoint subregions $\mathbb{T}_k$ and $\mathbb{T}_{k+1}$ such that $\mathbb{T}_k = \mathbb{T}_{2k} \cup \mathbb{T}_{2k+1}$, then the globally optimal solution must be contained in one or the other subregion. If it does not contain the globally optimal solution, then the region containing the globally optimal solution is still somewhere else in the priority queue and is unaffected by the operation. Therefore, if the priority queue contained a region containing the optimal solution $T_{\max}$ at the beginning of the loop and the algorithm does not terminate, then it must contain either the same or some other region containing the optimal solution at the end of the loop.

Now it remains to be demonstrated that when the algorithm terminates, it terminates with an optimal solution. When the algorithm terminates (in Step 4), observe that, by construction in the termination step, $Q(T_{\max}) = \hat{Q}(\mathbb{T}_k)$. Furthermore, because we are using $\hat{Q}$ as the priority in the priority queue, we immediately establish that $Q(T_{\max}) = \hat{Q}(\mathbb{T}_k) \geqslant \hat{Q}(\mathbb{T}_{\text{remaining}})$, where $\mathbb{T}_{\text{remaining}}$ is any remaining state in the priority queue. Therefore, further expansion of nodes from the priority queue cannot ever yield a solution that is better than $T_{\max}$.

Note that optimality here refers to the actual, finite precision numerical function Q, not some idealized mathematical counterpart. Global optimization of Q interpreted in terms of infinite precision real numbers but implemented using finite precision arithmetic is also possible, but it requires additional techniques and will be described in a separate paper.

Theorem 2. *Algorithm* 1 *terminates.*

For transformations represented using vectors finite precision numbers (integer or machine floating point), termination follows simply from the fact that there is only a finite number of points in transformation space. If we start with a region $\mathbb{T}$, each dimension can be split in half at most a finite number of times before reaching an elementary floating point interval. If the maximum number of splits possible before reaching an elementary floating point interval along any of d dimensions is b, then the maximum number of distinct transformation regions we can explore is 2^{bd}, and each such terminal region will contain only a single transformation. Because we require that $\hat{Q}(\{T\}) = Q(T)$, where T is the single remaining transformation in $\mathbb{T}$ and the termination condition in Step 4 is trivially satisfied.

Of course, in practice, subdividing transformation space until we arrive at a single transformation is usually unnecessary. Instead of terminating at the elementary floating point interval, we can terminate at some larger, chosen tolerance τ, that is, when $\operatorname{diam}(\mathbb{T}) < \tau$. This means that the algorithm finds the globally optimal solution in the d-dimensional discretized transformation space $\tau \cdot \mathbb{Z}^d \cap \mathbb{T}_{\text{all}} = \{\tau v | v \in \mathbb{Z}^d \wedge v \in \mathbb{T}_{\text{all}}\}$.

We can also think of such solutions as *weak solutions* [17] to the geometric matching problems in a continuous space of transformations. A weak solution is a solution that is not necessarily optimal but can become optimal under some small perturbation related to τ.

It is also useful to consider how we can compute the termination condition efficiently. In Step 4 of the algorithm, the termination condition is that the region in transformation space associated with the current state contains a transformation T such that $Q(T) = \hat{Q}(T)$. One simple way of picking such a transformation is to use the transformation T_c already used in the computation of $\hat{Q}(\mathbb{T})$. That is, we compute an upper bound $\hat{Q}(\mathbb{T})$ by evaluating, for each pair m, b on the match list $\lfloor \|T_c(m) - b\| < \epsilon + \delta \rfloor$, as well as $\lfloor \|T_c(m) - b\| < \epsilon \rfloor$. The computationally costly part of this evaluation, the transformation of the model point and the distance computation, is shared, so that this adds almost no cost to the overall computation.

Note that $Q(T)$ for $T \in \mathbb{T}$ is a lower bound on the maximum value of Q achievable over $\mathbb{T}$, giving us a range of values that any maxima in $\mathbb{T}$ might assume. In particular, if $\mathbb{T}_0$ and $\mathbb{T}_1$ are the two regions in transformation space corresponding to the top and second entries in the priority queue, and if $T_0 \in \mathbb{T}_0$, then seeing that $Q(T_0) > \hat{Q}(\mathbb{T}_1)$ tells us that $\mathbb{T}_0$ must contain a globally optimal solution.

While the choice for splitting regions in transformation space in Step 5 is the simplest to explain and will lead to eventual convergence, as a practical matter, there may be other useful choices. For example, we might want to choose the split that minimizes the uncertainty in the location of transformed model features. Or, we might split along multiple dimensions simultaneously.

Related to termination and splitting is the question of computational complexity. Determining theoretically the average and worst case complexity of these kinds of branch-and-bound matching algorithms remains an open problem. Heuristic arguments [7] and the empirical results presented below suggest that their average complexity is the same as that of alignment and that cases with worse complexity are rare, if they occur at all. The running time of the algorithm depends, among other things, on how well $\hat{Q}(\mathbb{T})$ approximates $\max_{T \in \mathbb{T}} Q(T)$ as $\operatorname{diam}(\mathbb{T}) \to 0$. A more formal theoretical analysis of these relations is clearly desirable.

5. Search strategy

So far, we have described a commonly used branch-and-bound strategy: expanding the most promising search node, no matter at what depth in the search tree it may occur. We might call this a "best first" search algorithm. An alternative approach is to use a "depth first" approach; this was the approach described in the first version of the RAST (recognition by adaptive subdivision of transformation space) algorithm [6,7]. The performance of best-first and depth-first strategies for geometric branch-and-bound matching have not been compared so far in the literature, leaving us uncertain about whether one or the other strategy might significantly outperform the other. In particular, we might be concerned that a depth-first search would be susceptible to "getting lost in regions of transformation space where no solution will ultimately be found, potentially resulting in very high running times compared to a best-first approach."

A depth-first search can be implemented by using search depth instead of the upper bound $\hat{Q}$ as the priority in the priority queue. When two nodes have the same depth, we prefer the node with the higher quality of match value. With this modification, we are no longer guaranteed anymore that the first solution that satisfies our acceptance criteria in Step 4 is, in fact optimal. The algorithm therefore needs to continue searching until the complete search tree has been examined.

At first sight, this may seem to result in a much larger number of node expansions and therefore might supposed to be considerably less efficient. What makes this algorithm practical is the observation that when the algorithm has found a candidate solution in Step 4, we need not expand further any states that we encounter whose upper bound estimate is less than, or equal to, the quality of the best solution we have found so far. If we modify the algorithm accordingly, this means that the additional cost in terms of runtime of a depth-first search strategy is often modest; furthermore, experimentally, the relative overhead becomes smaller the larger the geometric matching problem becomes (Fig. 2).

Why might we want to adopt a depth-first search strategy? Because the amount of memory it requires is much smaller than that of a best-first strategy. Over a common range of parameters explored in our experiments, the depth-first search approach requires between 1/50 and 1/200 the amount of memory of the best-first strategy (Fig. 3). This was, in fact, the motivation for choosing a depth-first approach in the branch-and-bound algorithm described in [6,7], since it allowed implementation on typical workstations at the time, which had 4–16 Mbytes of memory. Even today, the moderate increase in running times associated with a depth-first search may be justifiable in return for greatly reduced memory requirements when implementing these algorithms on embedded systems. A depth-first approach also makes it easier to tolerate the larger memory footprint associated with splitting regions $\mathbb{T}$ into 2^n subregions, rather than using binary splits in Step 5 in Algorithm 1, resulting in performance that can equal or surpass that of a best-first search with binary splits (data not shown).

A limitation of the depth-first search approach is that it does not permit us to find multiple distinct solutions to the matching problem using the incremental algorithm described in Section 8. Instead, the depth-first search must be run multiple times: first, to find the best solution, and then, to find the second best solution that is non-overlapping with the first one, on a modified problem from which the features participating in the match of the best solution have been removed, etc.

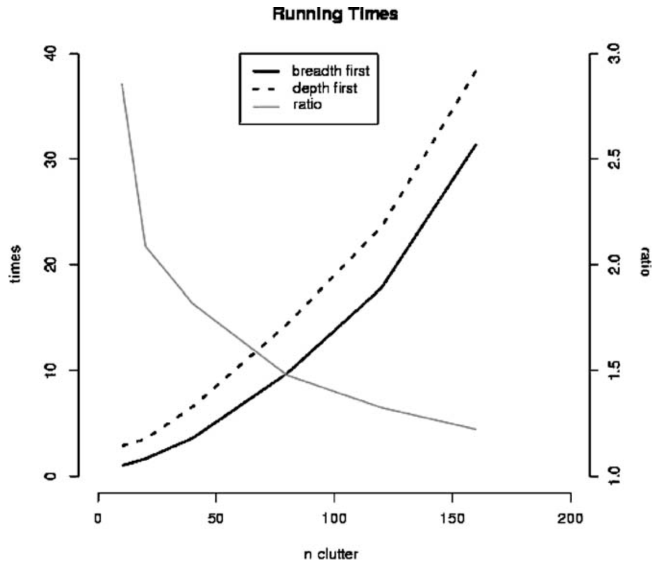

Fig. 2. Running times of the depth-first and best-first search strategies compared. The problem is as described in Section 3, with the number of clutter points given by the x-axis. The left y-axis shows the running time in seconds, the right y-axis shows the ratio of the depth-first running times to the best-first running times.

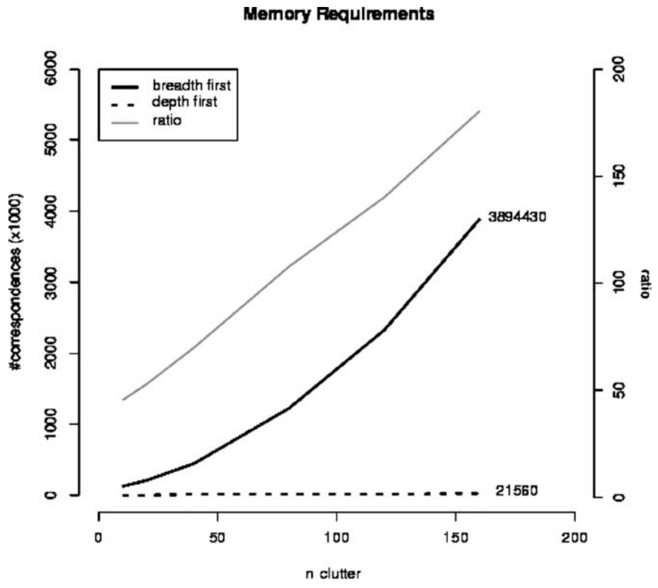

Fig. 3. Comparison of the memory requirements of the depth-first and best-first search strategies. The problem is as described in Section 3, with the number of clutter points given by the x-axis. The left y-axis shows the number of correspondences (in thousands) that need to be retained in the search queue (proportional to the memory requirements) seconds, the right y-axis shows the ratio of the best-first to the depth-first memory requirements.

6. Computation of upper bounds

In the discussion so far, there has been no mention of how to compute the upper bounds $\hat{Q}(\mathbb{T})$. The algorithm described in [6] takes advantage of special properties of linear error bounds under translation, rotation, and scale [2] or affine transformations that allow this test to be carried out easily and efficiently. This allows $\hat{Q}(\mathbb{T})$ to be determined using a small number of dot products and results in a tight upper bound.

For other kinds of transformations and error measures, a tight upper bound is difficult to compute, but we do not actually require a tight upper bound for the algorithm to converge. Therefore, branch-and-bound algorithms can be formulated using more general upper bounds that are easier to derive [9].

We start by determining a bound on the location of each model point m_j under any transformation $T \in \mathbb{T}$. This bound can be expressed as any convenient geometric region, for example a bounding rectangle or a bounding circle in the image plane. For the purpose of exposition, let us assume that we express this bound as a circular error bound of diameter δ around some point $T_c(m)$ for some $T_c \in \mathbb{T}$ (preferably "central" in that region). T_c and δ can, in general, be dependent on m. That is, we choose T_c and δ such that $\forall T \in \mathbb{T}: \ \|T_c(m) - T(m)\| \leqslant \delta$. Then it is easy to see (using the triangle inequality) that

$$\forall T \in \mathbb{T},\ b \in \mathbb{R}^2 : \|T(m) - b\| \leqslant \|T_c(m) - b\| + \delta. \tag{3}$$

Finally, we can use this to bound $Q(\mathbb{T})$. Let T be any transformation in some subregion $\mathbb{T}$ of transformation space. Then, by bounding the terms of the sum individually using Eq. (3):

$$Q(\mathbb{T}) = \max_{T\in\mathbb{T}} \sum_{m\in M} \max_{b\in B} \lfloor \|T(m) - b\| < \epsilon \rfloor \tag{4}$$

$$\leqslant \sum_{m\in M} \max_{b\in B} \lfloor \|T_c(m) - b\| < \epsilon + \delta \rfloor. \tag{5}$$

Given this inequality, we can compute $\hat{Q}(\mathbb{T})$ fairly simply by computing T_c and δ and evaluating the summation and maximization over all pairs of model and image features.

A simple optimization is to speed up the search for matching image points, that is, points b that fall within a distance of $\epsilon + \delta$ by using a point location data structure [26]. A particularly simple point location data structure is the distance transform, which has also been applied in branch-and-bound type geometric matching algorithms [4]. Both the use of point location data structures and the use of distance transforms can become fairly complicated to implement when features with several different geometric attributes or partial matches are involved. This paper presents a number of results showing that an approach that is simpler to implement, branch-and-bound matching based on *matchlists*, also results in better performance on many problems.

A matchlist-based approach works as follows. With each state in the priority queue in Algorithm 1, we associate not just a region in transformation space and a bound, but also a list of all the correspondences between model features and image features that are consistent with matching under the error bounds and the transformations represented by that region. As we subdivide regions in transformation space, we only need to examine correspondences that are consistent with the parent region (this fact requires a simple proof which is left to the reader). This approach based on *matchlists* was the approach introduced in [6].

7. Statistical properties of matchlists

A priori, it is not obvious that a matchlist-based approach is faster than an approach using point location data structures. A matchlist can have a size as large as the product of the number of model points and the number of image points, $|M|\,|B|$. The evaluation of $\hat{Q}(\mathbb{T})$ might therefore require as many as $|M|\,|B|$ steps using a matchlist approach. In contrast, point location algorithms can be implemented that run in nearly constant or logarithmic time in the size of the number of database points, suggesting a complexity nearly independent of the number of image points $|B|$ for the step of evaluating $\hat{Q}(\mathbb{T})$.

In fact, the initial matchlist in a matchlist-based approach is of size $|M|\,|B|$, and the initial few subdivisions do not reduce the matchlist at all. Only once $\mathbb{T}_k$ becomes small enough so that the uncertainty $\epsilon + \delta$ of the projected model features will become sufficiently small so as not to include all image features, will the match list be reduced below that size. The efficiency of a matchlist-based approach to evaluating $\hat{Q}(\mathbb{T})$ then depends on the size of the average matchlist that occurs during the search. If most search states are expanded with small matchlists, a matchlist-based approach may be efficient. If most search states were expanded with large matchlists, the matchlist-based approach would be hopelessly inefficient compared to approaches based on point location.

7.1. Average runtime behavior on randomly generated images

The question of the size of matchlists can be addressed by collecting statistics from actual runs of the algorithm. For the problem defined in Section 3, these results are shown in Fig. 4. As we can see, the average number of image features per model feature the matchlist $\bar{l}$ is a little over two, meaning that there are, on average, a little over two image features on the matchlist for each model feature. This makes it difficult to outperform the matchlist-based approach using a point location data structure. First, the matchlist-based approach only computes distances for model features that are still candidates for matching some image feature. Second, even if the number of model features considered by the two approaches were similar, the lookup performed by the point location data structure would probably have to take less time than the time for two computations of $\|m' - b\|$ (plus a small amount of bookkeeping overhead) in order for the overall implementation to run faster.

It is instructive to look at these statistics in a little more detail. Fig. 5 contains two types of curves. The solid curve indicates what fraction of the total number of search states that are expanded by the algorithm are expanded below a certain depth. We see that most of the states are expanded between depth 12 and 20. The broken line

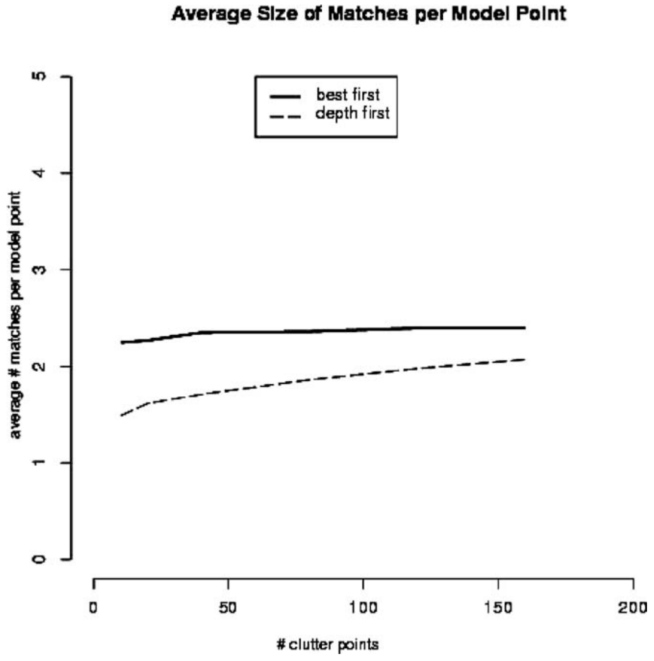

Fig. 4. Size of the average matchlist. The matching problem used is as given in Section 3. As in the other results, the x-axis gives the amount of clutter; the y-axis shows the number of correspondences on the average matchlist.

indicates the average number of image features per model feature; it drops sharply below depth 12, the depth at which most of the nodes are expanded.

There is a fairly simple heuristic explanation for this behavior. The algorithm must keep subdividing regions in transformation space recursively until it can start pruning the search tree, leading to an initial exponential growth in the number of nodes, However, pruning is possible only once the matchlist has shrunk significantly. Therefore, exponential growth of the number of nodes will tend to occur somewhat beyond the point where the matchlist has shrunk, resulting in the observed runtime behavior. The results in Fig. 6 suggest a nearly perfectly exponential growth up to a certain depth, followed by the onset of pruning.

7.2. Direct experimental comparison of point location approaches and matchlists

Results in the previous section, as well as results (not shown) from execution profiling already suggest that matchlist approaches might outperform point location based approaches. To test this, the performance advantage of the matchlist-based approach was also verified directly. Mount et al. [26] describe a branch-and-bound matching algorithm using a point location data structure. As their point location

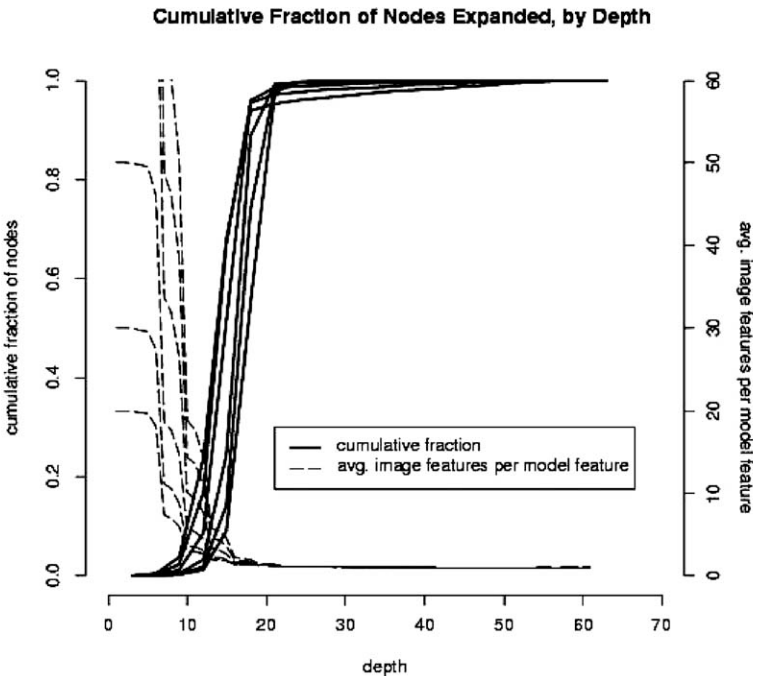

Fig. 5. Fraction of nodes expanded at different depths, and average number of image features matching each model feature at different depths. These results are for the problem described in Section 3; the different curves are at a clutter of 10, 20, 40, 80, 120, and 160.

data structure, they use the ANN library published and described by Mount and Arya [25]. This ANN code was used to conditionally replace the matchlist-based approach in the implementation of branch-and-bound matching used elsewhere in this paper.

This results in a controlled experiment comparing the two approaches. The only variable that differs between the two experimental conditions is whether matchlists or point location data structures are used in the computation of $\hat{Q}(\mathbb{T})$. All other aspects of the implementation, like the computation of transformations, the test dataset, and the priority queue implementation, are identical.

Using the point location data structure in place of the matchlist on the randomly generated problem instances resulted in a slowdown of a factor of 2.43 and 3.75 (for 120 and 10 points of clutter, respectively), confirming the prediction that the matchlist approach outperforms a point location-based approach under the experimental conditions tested.

7.3. Experiments with the COIL-20 image database

Features extracted from real images are not usually uniformly distributed. To confirm the results from the previous section, an additional set of experiments on

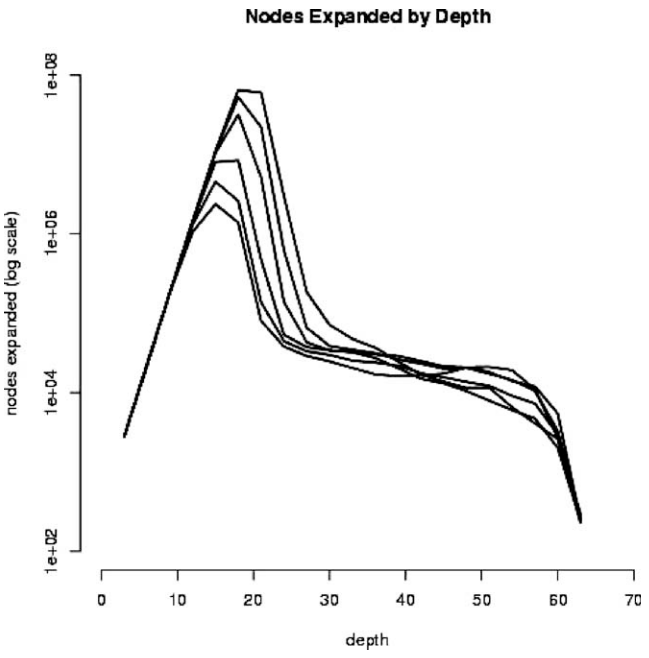

Fig. 6. Number of nodes expanded at different depths (logarithmic scale). Notice the initial exponential growth. These results are for the problem described in Section 3; the different curves are at a clutter of 10, 20, 40, 80, 120, and 160.

real image data was carried out. For this purpose, edges were extracted from images in the COIL-20 database [27] using a Canny edge detector. The resulting edges were approximated by polygons and sampled uniformly at 4 pixel intervals. This gives rise to collections of edge pixels consisting of between 82 and 283 edge samples for each image. Furthermore, four objects parts were chosen as model and processed in the same way, resulting in between 9 and 33 edge samples per model. The four models and a sample image are shown in Fig. 7. For reasons already outlined in Section 3, feature orientations were not used to speed up matching.

For the benchmarks, each of the four parts was then matched, using the point location and matchlist-based implementations, against the same set of 50 randomly selected images, resulting in a total of 200 runs of the algorithm. The running times of the two algorithms are shown in the scatter plot in Fig. 8, where each data point corresponds to one problem instance. The slope of a line fitted to that scatter plot is approximately 3.7. This reproduces the results obtained for uniformly random problem instances obtained on the random problem instances in the previous section.

In summary, these results suggest that matchlist-based implementations of branch-and-bound algorithms are likely preferable in many applications: they are fast, easy to implement, and make available a full set of candidate matches for the evaluation of the quality of match function $Q(T)$.

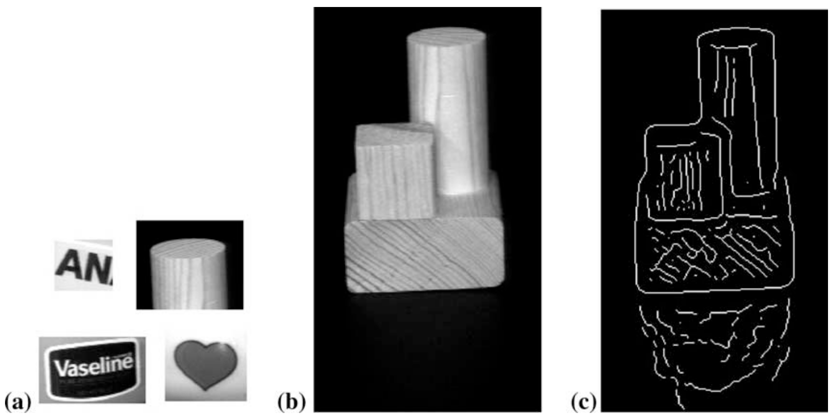

Fig. 7. Sample models and images from the COIL-20 database used in the performance measurements. Shown are (a) four models matched against the images, (b) a sample image, (c) edges extracted from the sample image and used by the matching algorithm.

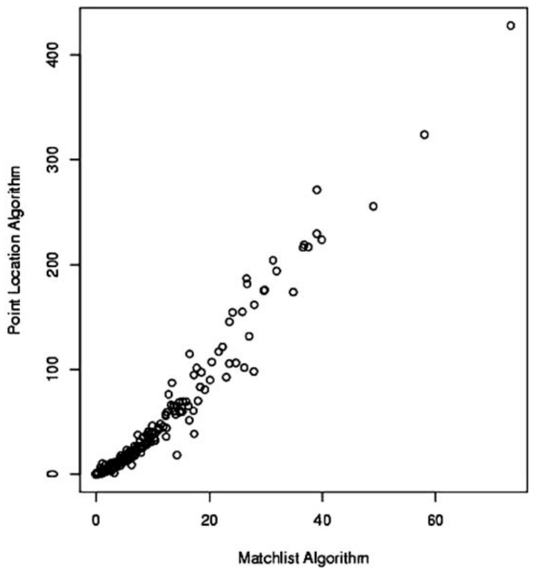

Fig. 8. Relative performance of the matchlist method compared to the point location method. Both axes show running times in seconds. Note the different scales of the axes. The slope of the curve is 3.7, meaning that the matchlist method runs on average 3.7 times faster than the point location method.

8. Multiple distinct matches

In many computer vision applications, an image may contain more than a single instance of an object. In Algorithm 1, after we have found the first match, rather than terminating, we can continue expanding nodes until we find the next solution. This will be the solution with the second highest quality.

However, while this is the second-best match, it usually does not represent a second, distinct instance of the model in the image, but rather a slight variation on the first match–the global optimum is usually surrounded by many similar near-global optima. Trying to exclude such near optimal matches based on a threshold on their distance in transformation space seems to be difficult: there is no a priori reason to assume that distinct instance of a model in an image are associated with significantly different viewing parameters. What distinguishes multiple instances of a model is that they are usually composed of separate sets of features.

One common approach to finding multiple model instances is that of a greedy match. In a greedy search strategy, we first find the best instance of the model in the image. Then, we remove all image features participating in the match from further consideration and re-start the search. This will yield a second match involving only features that were not already used in the top match. This can be repeated to find additional model instances. Such a greedy strategy can often be justified based on general viewpoint assumptions—under a general viewpoint, it is unlikely that multiple model instances share features.

With a simple modification of Algorithm 1, we can obtain multiple distinct instances more quickly and without restarting the search. For this, we introduce the notion of a weight w_b associated with each point b in our collection of image points. The function $Q(T)$ is then modified by the introduction of these weights

$$Q(T) = \sum_{m \in M} \max_{b \in B} w_b \lfloor \|T(m) - b\| < \epsilon \rfloor. \tag{6}$$

Initially, these weights are set to 1. When we have found the best match, we set the weights associated with all the matching image points to 0. The entries in the priority queue do not require updating since the previously computed upper bound estimates $\hat{Q}(\mathbb{T})$ are easily seen to be still upper bounds. That is, if $\hat{Q}'(\mathbb{T})$ is the upper bound after the update to the weights and $\hat{Q}(\mathbb{T})$ is the upper bound before the update, then $\hat{Q}(\mathbb{T}) \geqslant \hat{Q}'(\mathbb{T})$. Next time any search state is expanded, the upper bounds will use the new weights. This incremental approach works well in practice and has been used, for example, in applications of branch-and-bound algorithm to line finding [11] and geometric matching.

Note that the approach takes advantage of the use of matchlists; if point location data structures or discrete Voronoi diagrams are used in the computation of $\hat{Q}(\mathbb{T})$, updating the state of the search to disregard those image features that have previously participated in a match appears to be a much harder problem because the point location data structures would have to be updated efficiently dynamically.

9. Other quality-of-match functions

9.1. Feature orientations

In many practical applications, point features carry some kind of information about orientation. For example, samples from the edges found in an image are associated with gradient direction information. As already noted in Section 3, using orientation information can greatly speed up matching, as well as reduce the probability of false positive matches. We can incorporate this information into the matching process by requiring that points that match are not only mapped to within a certain error bound by the transformation, but also that their associated directions are mapped to within a certain angle of each other. This adds a factor to the quality of match function (we use $|x - y|_\pi$ to denote the normalized difference in angle, $\min_{k\in\mathbb{Z}} |x - y + k\pi|$):

$$Q(T) = \sum_{m\in M} \max_{b\in B} \lfloor \|T(m) - b\| < \epsilon \rfloor \cdot \lfloor |T(\beta_m) - \beta_b|_\pi < \epsilon \rfloor. \tag{7}$$

Orientations are a simple example of properties that change under transformation of model features into the image plane, but in ways different from location. These kinds of properties are particularly easy to handle in a matchlist approach [12] because the only modification to the algorithm that is necessary is to change the quality of match function to incorporate the additional factors. Furthermore, in a matchlist approach, these additional constraints will serve immediately to prune the search tree. Incorporating these kinds of features into a point location framework tends to be more difficult because it requires the use of point-location data structures over higher-dimensional spaces with topologies different from $\mathbb{R}^n$.

9.2. Gaussian error and MAP estimates

The same algorithm described above works essentially unchanged for a Gaussian error model rather than a bounded error model. In the simplest case of Gaussian error, a fixed Gaussian associated with each image feature, a commonly used likelihood function is given by [22]:

$$L(T) = \prod_{m\in M} \max_{b\in B} \max(G_{\sigma I}(T(m) - b), P_0). \tag{8}$$

Here, σI is a simple diagonal covariance matrix and P_0 is associated with the probability of having a feature point appear in the image that was not derived from an instance of a model (the ‘‘probability of clutter’’). A MAP (*maximum a posteriori*) estimate of the transformation T is a function that maximizes this likelihood.

Taking logarithms on both sides, rescaling, and introducing a constant ϕ_0, we obtain a quality of match function $Q(T)$ with the same maxima as the likelihood function $L(T)$ and in a more convenient form:

$$Q(T) = \sum_{m\in M} \max_{b\in B} \max(-\phi_\sigma \|T(m) - b\|^2, -\phi_0). \tag{9}$$

Here, $\phi_0 = \log P_0$ and $\phi_\sigma = 1/2\sigma^2$. We can also rewrite it a little further to maintain the property that correspondences whose contributions to the quality of match function $Q(T)$ drops to zero can be removed from the matchlist, in analogy to the bounded error condition

$$Q(T) = \sum_{m\in M} \max_{b\in B} \max(-\phi_\sigma \|T(m) - b\|^2 + \phi_0, 0). \tag{10}$$

A Gaussian error model for angles can be introduced in a way analogous to Eq. (7):

$$Q(T) = \sum_{m \in M} \max_{b \in B} \max(-\phi_\sigma \|T(m) - b\|^2 - \phi_{\sigma'} |T(\beta_m) - \beta_b|_\pi^2 + \phi_0, 0). \tag{11}$$

The question remains of how to compute the bound $\hat{Q}(\mathbb{T})$. This, too, is simple. As before, we pick a transformation T_c and a bound δ for $\mathbb{T}$ and obtain

$$\hat{Q}(\mathbb{T}) = \sum_{m \in M} \max_{b \in B} \max(-\phi_\sigma \max(\|T_c(m) - b\| - \delta, 0)^2 + \phi_0, 0). \tag{12}$$

As already mentioned above, such a change in quality of match function affects our termination condition, since it will almost never be the case that $\hat{Q}(\mathbb{T}) = Q(T)$ for some T chosen at random from $\mathbb{T}$, no matter how small we make the region $\mathbb{T}$. However, this is nothing unusual: numerical optimizations in general can only return an accuracy to within machine precision or some lower, chosen tolerance. A branch and bound approach to finding a solution to geometric matching problems is simply such a numerical optimization algorithm. For a given tolerance τ, we can look at this kind of problem as replacing the original space of transformations $\mathbb{T}_{\text{all}}$ with a discretized space $\{\tau \cdot v : v \in \mathbb{Z}^3\} \cap \mathbb{T}_{\text{all}} = \tau\mathbb{Z}^3 \cap \mathbb{T}_{\text{all}}$ and finding the optimal solution with that discretized space.

Note that Jurie [23] describes a branch-and-bound algorithm for recognition under Gaussian error that relies on a more complicated local numerical optimization step and uses an alignment method to identify regions over which to perform local search. Olson [29] also has recently described a similar approach using a Voronoi diagram, but has to rely on a separate interpolation step to achieve subpixel accuracy. We will return to a comparison of a combined alignment and local search approach with a global branch-and-bound approach in the discussion.

9.3. Hausdorff matching

Bounded error matching computes the largest number of matches compatible with a given error bound. We can solve the complementary optimization problem of minimizing the error bound under which some transformation can bring a given number of model points into correspondence with some image point. This similarity measure is now commonly referred to as the *partial directed Hausdorff distance* [19,30]. Unlike bounded error matching, which has a natural statistical interpretation in terms of probabilistic location error and a probabilistic occlusion model [22], it is unclear whether there is a similarly natural statistical interpretation of the partial directed Hausdorff distance.

We could apply the framework described in this paper directly to the computation of partial directed Hausdorff matches by formulating a quality of match function $Q(T)$ for the Hausdorff distance. However, that is a fairly inefficient approach, and the algorithms described in [20,30] may be a better choice for such such computations. Another alternative is to take an algorithm for bounded error matching and transform it into a partial directed Hausdorff matching algorithm by a binary search over error bounds [7].

Recently, some authors have introduced the *directed fractional Hausdorff distance* [28]. Since this function is identical to the bounded error measure except for a normalization factor [28], the standard bounded error recognition quality of match functions discussed elsewhere in this paper apply.

9.4. Line segment features

There are a wide variety of plausible quality of match functions involving models and images that are represented as collections of line segments rather than point features (cf. the work by Grimson [16] for heuristic search methods). We can, for example, view a collection of line segments simply as a compact representation of a discrete or continuous set of points, paying no attention otherwise to the grouping of points implied by which points are assigned to which line segments. At the other end of the spectrum of possibilities, we can consider line segments as meaningful entities in themselves; for example, in an idealized line drawing of a 3D scene composed of wireframe polyhedra, there is a one-to-one correspondence between line segments and unoccluded edges of the 3D polyhedra. These different possibilities imply different quality of match functions. In particular, they affect how partial matches between line segments are apportioned. The discussion below describes a simple quality of match measure for line segments that appears to work well in many practical applications and is easy to compute.

In addition to being a useful model in its own right, the purpose of this section is to demonstrate branch-and-bound matching for a quality of match function $Q(T)$ that involves complicated, transform-dependent error bounds.

A line segment is a pair of points $(p,q) \in \mathbb{R}^2 \times \mathbb{R}^2 = \mathbb{R}^4$, and it transforms like $T((p,q)) = (T(p), T(q))$. We consider the pair unordered and therefore view (p,q) as the same line segment as (q,p). The model is a collection of line segments $M = \{m_1, \ldots, m_r\} \subseteq \mathbb{R}^4$ and the image is a collection of line segments $B = \{b_1, \ldots, b_s\} \subseteq \mathbb{R}^4$. The quality of match between a transformed model line segment $T(m) = (u,v)$ and an image line segment $b = (p,q)$ is defined as:

$$q(T(m), b) = f(\alpha(T(m), b)) \cdot g(\mathrm{dist}(p, \ell_{u,v})) \cdot g(\mathrm{dist}(q, \ell_{u,v})) \cdot \mathrm{overlap}(T(m), b). \tag{13}$$

The total quality of match $Q(T)$ used in the experiments below is

$$Q(T) = \sum_{m \in M} \sum_{b \in B} q(T(m), b). \tag{14}$$

Here, $\alpha(T(m), b)$ is the angle between the two line segments. The term $\ell_{u,v}$ is the line through u and v, and $\mathrm{dist}(p, \ell_{u,v})$ is the distance of p from that line. The term $\mathrm{overlap}(T(m), b)$ is the length of b when projected onto the line segment $T(m)$ along a direction perpendicular to $T(m)$. f is a function penalizing differences in orientation between the two line segments, and g is a function penalizing location error between the two line segments. For bounded error matching of line segments, we choose $f(\Delta\alpha) = \lfloor |\Delta\alpha| < \epsilon_\alpha \rfloor$ and $g(d) = \lfloor d < \epsilon \rfloor$. For approximate robust least square error matching[2] we choose $f(\Delta\alpha) = \max(0, 1 - \epsilon_\alpha^{-2}|\Delta\alpha|^2)$ and $g(d) = \max(0, 1 - \epsilon^{-2}d^2)$. Computation of the upper bounds $\hat{Q}(\mathbb{T})$ in both cases is carried out by incorporating the δ parameter into these equations in a way analogous to point features.

It is interesting to compare this quality of match function with the one described by Yi and Camps [31]. Yi and Camps treat line segments essentially as rigid bodies, parameterized by their center, the logarithm of their length, and their orientation; errors are measured in terms of the displacement of the center, plus two additional properties. This parameterization allows them to transform the line matching problem into a four-dimensional point matching problem under translation, solved using a generalization of the discrete Voronoi diagram approach described in [19]. One of the properties of this special quality of match function is that the shape of the error bounds it implies on location are invariant under rotation.

However, it is unclear that this is a desirable model in many applications. For example, a half-occluded line segment in the Yi and Camps model would incur a substantial location error and might be considered non-matching altogether. In contrast, the error model presented in this section allows for partial matches and lets us express quality of match functions in general ways in terms of orientation, proximity, and overlap between model and image line segments; a half-occluded line segment in the correct location would incur no location error.

If we expressed such a partial match error model in terms of a location error bound on the center of a model line segment, the error bound would be elongated along the line segment and this error bound would rotate as the model is transformed. This is an instance of a transformation-dependent error bound and appears to be difficult to address efficiently without the use of matchlists. The matchlist-based approach method in this paper is probably the first method for finding globally optimal solutions under such error bounds. Experimental results on the LiME dataset of hard test cases [3] are presented in Section 13.

10. Common transformation spaces and their bounds

In the previous section, we discussed extensively different quality of match functions we might want to utilize in geometric matching problems. In this section, we look at the other major ingredient to geometric matching problems: the space of transformations we are considering and the bounding functions that correspond to it.

The derivation of bounding functions corresponding to transformations is not conceptually hard, but algebraic mistakes are easy to make. The bounding functions described in this section have been verified numerically using random sampling, and

[2] This is only an approximation to the least square matching of line segments considered as point sets, as used in, for example, [3], but it is easier to compute and has some qualitatively desirable properties. A full discussion of the relative merits of different quality of match functions for line segment features is beyond the scope of this paper.

they are believed to be correct up to clerical errors.[3] The author hopes that presenting this collection of numerically verified bounding functions in a single place will help ease the implementation of branch-and-bound methods. No claim is made for the novelty of these functions; bounds like these have been derived by many authors for a variety of purposes.

Let us recall briefly the connection between quality of match functions and transformation space. In order to apply the algorithms presented in this paper, we need to obtain an upper bound $\hat{Q}(\mathbb{T}) \geqslant \max_{T\in\mathbb{T}} Q(T)$. We have generally expressed the connection between the two by deriving, given a set of transformations $\mathbb{T}$, a representative transformation T_c and an error bound δ such that we can choose $\hat{Q}(\mathbb{T};\epsilon) = Q(T_c;\epsilon+\delta)$. Furthermore, the only sets of transformations $\mathbb{T}$ encountered by the algorithm are axis aligned hyper-rectangles in transformation space $\mathbb{T}_{\text{all}}$.

A different way of looking at this is in terms of the region that a model point m traces out under transformations in the set $\mathbb{T}$. Hagedoorn and Veltkamp [18] refer to this area as the *swept area*. Formally, this set is written as as $\{Tm : T \in \mathbb{T}\} \in \mathbb{R}^2$. We are bounding this set by the circle centered at T_c and with radius $\epsilon+\delta$. Of course, the swept area can be bounded in other ways as well; rectangular bounds are convenient in many applications. For the purposes of this paper, we will restrict ourselves mostly to circular bounds. Unless otherwise mentioned, all transformations will be from $\mathbb{R}^2 \to \mathbb{R}^2$.

10.1. Translation

Let us parameterize the transformation by the x displacement t_x and the y displacement t_y. That is,

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{15}$$

Let $\mathbb{T} = [t_x^0, t_x^1] \times [t_y^0, t_y^1]$, and let $t_x^m = \frac{1}{2}(t_x^0 + t_x^1)$, $t_x^d = \frac{1}{2}(t_x^1 - t_x^0)$, $t_x^D = t_x^1 - t_x^0$, and equivalently for t_y. In fact, for implementing a branch-and-bound matching algorithm under translation, in this particular case, it is easiest to use this bound on coordinates, rather than a circular bound. The swept area of a model point $m = (x,y)^{\mathrm{T}}$ is then simply the rectangle $[x+t_x^0, x+t_x^1] \times [y+t_y^0, y+t_y^1]$. Let us use the following notation for expressing this rectangular error bound on the coordinates of a transformed model point:

$$T_c = (t_x^m, t_y^m)^{\mathrm{T}}, \tag{16}$$

$$t_x^\delta = \frac{1}{2}(t_x^1 - t_x^0) = t_x^d, \tag{17}$$

$$t_y^\delta = \frac{1}{2}(t_y^1 - t_y^0) = t_y^d. \tag{18}$$

[3] Please communicate any corrections to the author.

However, the circular bound is easily derived as well. We choose

$$T_c = (t_x^m, t_y^m), \tag{19}$$

$$\delta = \sqrt{(t_x^d)^2 + (t_y^d)^2}. \tag{20}$$

10.2. Translation and scaling

Let us parameterize the transformation $T = (t_x, t_y, s)^{\mathrm{T}}$ such that

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = s\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{21}$$

Using the same conventions for superscripts as in the previous section, we choose

$$T_c = (t_x^m, t_y^m, s^m), \tag{22}$$

$$\delta = \sqrt{(t_x^d)^2 + (t_y^d)^2} + s^D |m|. \tag{23}$$

Here, $|m|$ is the magnitude of the vector representing the position of the model point; we can replace this with $\max_{m \in M} |m|$ for a less tight bound, albeit at a significant decrease in runtime performance.

10.3. Translation and non-uniform scaling

Let us parameterize the transformation $T = (t_x, t_y, s_x, s_y)^{\mathrm{T}}$ such that

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} s_x x \\ s_y y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{24}$$

We choose

$$T_c = (t_x^m, t_y^m, s_x^m, s_y^m), \tag{25}$$

$$\delta = \sqrt{(t_x^d)^2 + (t_x^d)^2} + \sqrt{(s_x^D m_x)^2 + (s_y^D m_y)^2}. \tag{26}$$

Here, m_x and m_y are the coordinates of the model point being transformed; as before, we can replace these with upper bounds over the set M.

10.4. Isometric transformations

Isometric transformations are translations and rotations, without scale changes. Let us parameterize the transformation $T = (t_x, t_y, \alpha)^{\mathrm{T}}$ such that

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{27}$$

Here, α is in radians. We choose

$$T_c = (t_x^m, t_y^m, \alpha^m), \tag{28}$$

$$\delta = \sqrt{(t_x^d)^2 + (t_x^d)^2} + \alpha^d |m|. \tag{29}$$

10.5. Isoform transformations, linear parameterization

Isoform transformations are 2D translations, rotations, and uniform scaling. Let us parameterize the transformation $T = (t_x, t_y, c, s)^{\mathrm{T}}$ such that

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{30}$$

Note that $(c, s)^{\mathrm{T}}$ is related to the angle of rotation α and scaling λ as

$$\begin{pmatrix} c \\ s \end{pmatrix} = \lambda \begin{pmatrix} \cos\alpha \\ \sin\alpha \end{pmatrix}. \tag{31}$$

This case is actually less interesting for its bounds (the parameterization is somewhat inconvenient) but more for the fact that it allows convex polygonal error bounds in the image to be translated into convex polygonal error bounds in transformation space. For more details, see [2,7].

However, for completeness, here are the bounds. First, rectangular bounds using the notation introduced above for translations:

$$T_c = (t_x^m, t_y^m, c^m, s^m), \tag{32}$$

$$t_x^\delta = c^d |m_x| + s^d |m_y| + t_x^d, \tag{33}$$

$$t_y^\delta = s^d |m_x| + c^d |m_y| + t_y^d. \tag{34}$$

In circular error bounds, this becomes

$$T_c = (t_x^m, t_y^m, c^m, s^m), \tag{35}$$

$$\delta = \sqrt{(c^d |m_x| + s^d |m_y| + t_x^d)^2 + (s^d |m_x| + c^d |m_y| + t_y^d)^2}. \tag{36}$$

10.6. Isoform transformations, nonlinear parameterization

Let us use a more convenient parameterization of isoform transformations as $T = (t_x, t_y, \alpha, s)^{\mathrm{T}}$, $s > 0$, such that

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = s \begin{pmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{37}$$

The corresponding bounds are

$$T_c = (t_x^m, t_y^m, \alpha^m, s^m), \tag{38}$$

$$\delta = \sqrt{(t_x^d)^2 + (t_y^d)^2} + s^d|m| + \alpha^d s^1 |m|. \tag{39}$$

10.7. Affine transformations, linear parameterization

We define affine transformations as

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{40}$$

Again, first, here are the rectangular error bounds:

$$T_c = (t_x^m, t_y^m, c^m, s^m), \tag{41}$$

$$t_x^\delta = a^d|m_x| + b^d|m_y| + t_x^d, \tag{42}$$

$$t_y^\delta = c^d|m_x| + d^d|m_y| + t_y^d. \tag{43}$$

In circular error bounds, this becomes:

$$T_c = (t_x^m, t_y^m, c^m, s^m), \tag{44}$$

$$\delta = \sqrt{(a^d|m_x| + b^d|m_y| + t_x^d)^2 + (c^d|m_x| + d^d|m_y| + t_y^d)^2}. \tag{45}$$

10.8. 3D recognition

There is a wide variety of 3D imaging transforms possible. A direct application of those models to the recognition of point features is often impractical because the complexity is too high; efficiency requires the use of more complex features: line segments, corners, etc. A complete enumeration of the possibilities goes beyond the scope of this paper.

To illustrate, let us describe a simple 3D imaging transform: orthogonal projection under rotations out of the image plane and translations. We leave the derivation of bounds for other cases to the reader. The imaging transform then becomes (with the z axis pointing towards the camera):

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix}. \tag{46}$$

The bounds for this transformation become

$$T_c = (t_x^m, t_y^m, \alpha^m, s^m), \tag{47}$$

$$\delta = \sqrt{(t_x^d)^2 + (t_y^d)^2} + \alpha^d|m| + \beta^d|m|. \tag{48}$$

11. Significance of optimality

It is worth asking whether the optimality guarantees made by these branch-and-bound methods actually make a measurable difference compared to the matches found by commonly used approximations like alignment. If even simple, current methods for geometric matching were to return optimal matches with high probability, then the optimality guarantees made by the algorithms presented in this paper would not be particularly interesting.

The purpose of the empirical comparisons below is not to show that optimal geometric matching is better than any other, possibly heuristic, method that has ever been described, it is to compare the performance of optimal geometric matching to a widely understood baseline. Since most vision researchers probably either have implemented a simple alignment algorithm or a simple Hough transform algorithm, we compare the quality of the results from optimal geometric matching to the quality of results obtained by implementations of those algorithms.

For the implementation of geometric matching by alignment [21] used in this comparison, every possible pair of image features was put into correspondence with every possible pair of model features. For each such correspondence, the alignment transformation was computed and the resulting quality of match under the error bound was determined. That is, for alignment, we compute

$$\hat{Q}_a = \max_{T \in \mathbb{T}_a} \sum_{m \in M} \max_{b \in B} \lfloor \|T(m) - b\| < \epsilon \rfloor, \tag{49}$$

where $\mathbb{T}_a$ is the set of alignment transformations.

It is well known that geometric matching by alignment does not always result in geometrically optimal matches. The reason is that, in general, the optimal transformation does not equal any of the transformations computed from alignments. Alignment algorithms of any form can find some of the missing optimal matches is by exploring additional transformations around the alignment transformations using gradient descent or other local search methods (e.g. [3,18,23]). Another approach to addressing the limitations of simple alignment methods is the error propagation approach described by, e.g., Alter and Jacobs [1], which allows us to identify a set of potentially matching image features from an initial alignment; but without additional mechanisms, even alignment using such precise error propagation still does not identify the globally optimal bounded error match unambiguously (see also Section 14). Modifications to alignment algorithms that restrict the set of correspondences to be considered, for example, based on grouping information or the propagation of geometric error bounds can only result in a decrease of the overall quality of match because they will maximize the quality of match over a smaller set of transformations. So, while there are improvements to alignment methods, since alignment without local search or error propagation is widely used and accepted, it is interesting to see with what frequency optimal bounded error matching finds matches that are not represented by any alignment transformation.

Fig. 9 shows the difference between the number of features returned by an implementation of an optimal algorithm based on the techniques described in this paper

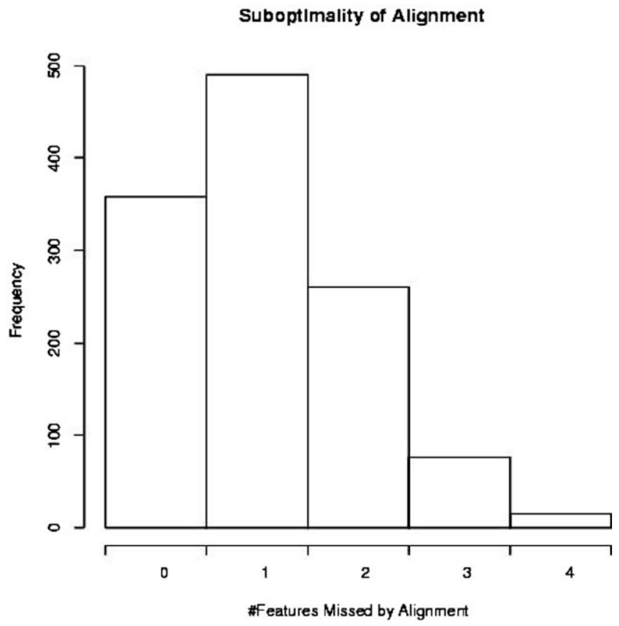

Fig. 9. Histogram of the number of matching image features missed by alignment compared to optimal matching (histogram of 1200 trials, with clutter from between 10 and 160 points).

and the number of features returned by an alignment algorithm (for a correct implementation of an optimal algorithm, this difference can, of course, never be less than zero). The distribution is over the randomized trials as described in Section 3. These results show that the alignment algorithm actually returns a suboptimal solution in the majority of the randomized trials, and that on a significant fraction of the cases, the differences are substantial (recall that the model instance represents 10 points in the image, although by chance, an optimal match may include additional points in some trials).

Analogous experiments were carried out with a randomized Hough transform (RHT) [24]. To ensure maximum detectability with the RHT and reduce noise, the implementation did not only just sample pairs of image and model features, but accumulated over all possible correspondences. Furthermore, the RHT implementation allowed for the addition of jitter to the location of model features, allowing a stochastic approximation of the actual probability distribution of transformations in the space of Hough accumulators.

The Hough transform is intrinsically limited in its performance because for any practical quantization of the transform space, the quantization results in only a rough estimate of a transformation. Additional techniques, like numerical optimization or further hierarchical decomposition, might be used to determine the optimal solution after the first Hough transform step completes. However, for such additional techniques to succeed, at a minimum, the candidate bin ought to be the correct one. For the randomized trials as presented above, using a bucket size of $\epsilon/2$ (where ϵ is the error bound), in 83.7% of the cases, not even the translational component determined by the Hough transform was within 2ϵ of the correct value, making it unlikely that post-processing would be able to recover the correct transformation. Similar results were obtained for other bucket sizes and using jitter (the variety of possible parameter choices and their unpredictable effect on performance of the Hough transform is itself a disadvantage of the technique). Some of the limitations of Hough transform techniques result from their implicit choice of quality of match measure relative to geometric matching approaches. In particular Hough transform methods use as a quality of match measure the total number of correspondences between images and models consistent with some set of transformations; see [9] for an additional discussion.

Of course, there is a large number of heuristic techniques and parameter choices that can be used to make Hough transform methods work better in practice on specific problems. However, branch-and-bound matching, in comparison, returns optimal geometric matches with essentially no further tuning required. In fact, the techniques described in this paper can be viewed as a reliable way of implementing a hierarchical Hough transform; for a discussion of the relationship between the two approaches, see [9].

12. Empirical complexity

To get some idea of the scaling behavior of the branch-and-bound methods, their performance was evaluated relative to an optimized implementation of alignment on the problem of matching randomly generated point sets under translation and rotation, but constant scale. The goal is not to prove that the branch-and-bound algorithm is faster than alignment, but to understand its scaling behavior relative to a well-understood and easily reproducible method for geometric matching.

As noted in the previous section, for recognition by alignment, we pick two points from the model, two points from the image, and compute a transformation that aligns the two model points with the two image points. Using that transformation, we project the remaining model points into the image and count the number of matching features under the given error bound. In the alignment algorithm implemented for this benchmark, two point location data structures were used to speed up the matching algorithm (and it was verified experimentally that use of these data structures indeed results in a substantial speedup).

First, because of the absence of scale changes, only pairs of image features within a range of distances similar to that of the model features are candidates for alignments. These pairs are found quickly using a one-dimensional interval query data structure. Second, when the model points are projected into the image, image points that fall within the given error bound of some model point are found quickly using a two-dimensional point location data structure.

This implementation of alignment was compared on a set of 200 randomly generated trials to globally optimal branch-and-bound matching using matchlists,

nclutter	alignment	branch-and-bound	speedup
50	0.676	0.62	1.08
100	2.16	1.8	1.20
150	4.56	3.5	1.30
200	7.87	6.1	1.29

Fig. 10. Running times on 2D matching task. The table shows a comparison of running times of an alignment algorithm and an optimal branch-and-bound matching algorithm. The running times are given in seconds and represent averages of 200 trials. "nclutter" is the number of random, non-model-derived background features.

as described above. Experiments were carried out on a 1 GHz desktop PC. Fig. 10 shows the relative performance of the alignment algorithm and the branch-and-bound algorithm. Under the conditions tested, the branch-and-bound algorithm scales approximately the same way as the alignment algorithm and, in fact, has a slightly larger advantage for larger problem instances. These results are in agreement with a heuristic theoretical analysis presented previously [7,6]. The results suggest that, at least for a certain range of parameters and problems, branch-and-bound algorithms have similar runtime complexities and constants compared to alignment methods.

13. Partial line segment matching on the lime dataset

The results presented in this paper so far have been mostly on randomly generated point sets. We already noted the advantages of using that kind of data: it matches the statistical assumptions justifying many quality of match functions, it is easy to reproduce, and large amounts of data are easy to generate.

Beveridge and Riseman [3] argue that test cases involving a high degree of symmetry are considerably harder than such randomly generated data, and that it is important to test geometric matching and visual object recognition algorithms on such harder problem instances. This idea is supported by their practical experience with matching different kinds of models in the Line Matching Environment (LiME) [3], and they provide a set of hard test cases for line matching that involve various forms of symmetry.

In this section, we see the results of applying the branch-and-bound methods of this paper to those hard test cases. This is a stress test in that not only is the dataset constructed to be especially hard, but also that, as we observed in Section 9.4, the error model itself—partial line segment matching—is considerably more complex than point matching, since it involves transformation-dependent, non-circular error bounds. In fact, the techniques described in this paper may be the only known and implemented techniques for finding globally optimal solutions to such problems to date.

The purpose of this test is foremost to make sure that the algorithm does not behave unreasonably on those cases. Furthermore, while the dataset contains only

48 model/image pairs and is not large enough to explore independently all the different parameters that influence recognition performance, we can draw some simple qualitative conclusions about the effects of instance size and symmetry on running times. Finally, the timing results on this data allow the reader to get some additional idea of how well the algorithms perform on commonly used line segment data with non-uniform distributions (see Figs. 11 and 12).

The running times of a matchlist-based branch-and-bound algorithm using the quality of match functions for line segments described in Section 9.4 are shown in Fig. 13 (these were carried out on a 1GHz PC). The first observation is that robust least square matching (for these parameter settings) usually takes about 2–3 times as long as bounded error matching. This is similar to the overhead of robust least square matching compared to bounded error matching in other conditions, and it is largely due to the fact that a bounded error match can often terminate the search early when it discovers that $\hat{Q}(\mathbb{T}) = Q(T)$ for some $T \in \mathbb{T}$, while a robust least square match must continue optimizing T until a desired numerical accuracy is reached.

Comparing the performance on different kinds of models, we see that the "dandelion" model is the slowest to match, probably for a combination of reasons. First, it exhibits a high degree of symmetry, so that there are many near-optimal matches. Furthermore, the model contains many lines which are fairly close to one another, meaning that model line segments can be brought into unique correspondence with image line segments only when the uncertainty in the transformation (and hence the δ value) is small (a similar effect is likely also at work for the "pole" model). For a similar reason, spatially smaller models probably take more time to match than larger models, even if they are composed of similar numbers of line segments.

While it was not the primary purpose of conducting these measurements, we can also attempt to compare the absolute running times presented by Beveridge and Riseman in [3] with the running times in this paper. In order to do this, we need to take into account that they were carried out on different hardware. The system used in those measurements was a SPARC 10, while the system used for the measurements in this paper were carried out on a 1 GHz PC. The ratio of clock speeds of the two systems is about 25 (this appears to correspond also to the ratio of SPEC benchmark scores). The branch-and-bound algorithms, however, outperform the local search based methods by a factor of 400 on some of the harder benchmarks (e.g., the "da.4" model), suggesting that branch and bound methods might be computing globally optimal solutions faster than local search methods compute approximate solutions.

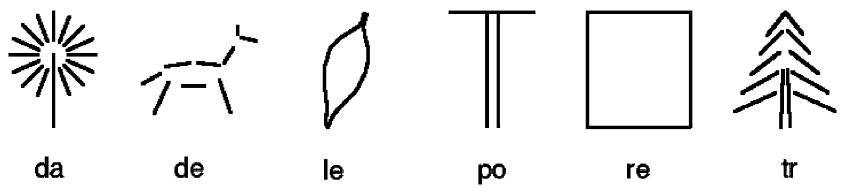

Fig. 11. The set of models used in the line segment matching experiments (see [3]).

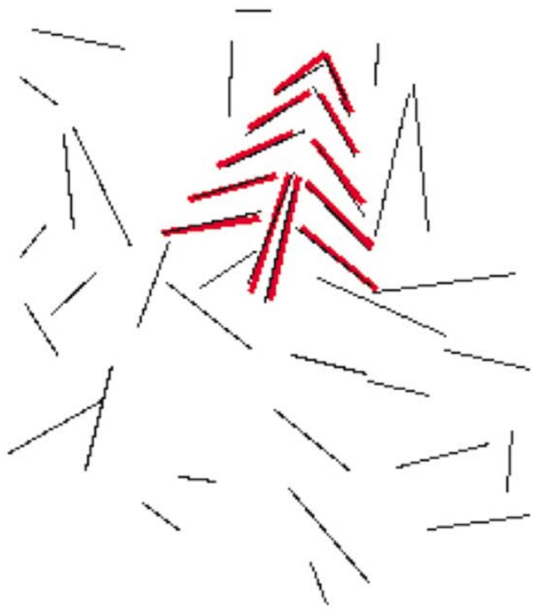

Fig. 12. Sample match of the “tr” (“tree”) model in a cluttered scene with 30 line segments. This corresponds to the running times labeled “tr30” in Fig. 13.

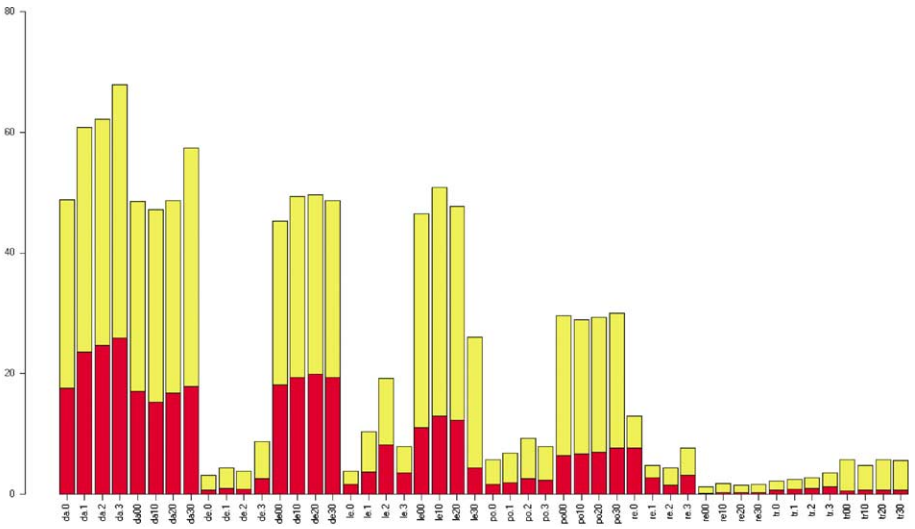

Fig. 13. Running times of the line segment based branch-and-bound algorithm applied to the LiME dataset [3]. Running times are in seconds. Dark bars represent bounded error matching with $\epsilon = 3$ and light bars represent Gaussian error matching with $\epsilon = 6$. Key: “da”—“dandelion”; “de”—“deer”; “le”—“leaf”; “po”—“pole”; “re”—“rectangle”; “tr”—“tree”; “.*n*”—image with *n* suboptimal model instances as clutter; “*nn*”—image with *nn* line segments of random clutter.

Independent of hardware, we can also compare sensitivity of the two methods to multiple model instances and symmetry by looking at ratios of running times under different conditions. Without wanting to claim statistical relevance, let us look at two examples. First, we find that for the "dandelion" model, the running times for between one and four model instances vary by a factor of 43 and 91 for the two matching methods described in [3]. In comparison, running times for finding the global optimal match using branch-and-bound methods under the same conditions varies only by a factor of 2.2. In terms of dependence on symmetry, if we compute the ratio of running times for a model with a lot of symmetry, like the "dandelion" model, to running times for a model with comparatively little symmetry, like the "leaf" model, we find that the local search methods and branch-and-bound methods are comparable: the highly symmetric model takes approximately ten times as long to match as the less symmetric model.

As mentioned at the beginning of this section, the purpose of this section was to test whether branch-and-bound algorithms for geometric matching behave reasonably on a set of previously studied hard test cases. Application to this dataset of hard test cases has failed to turn up any conditions to which branch-and-bound methods are unduly sensitive; if anything, there are indications that branch-and-bound methods may be somewhat less sensitive to multiple model instances than local search methods. While the dataset is small, the results also suggest that the performance of branch-and-bound algorithms may compare favorably to local search methods. When the community has settled on a larger set of hard test cases, it would be useful to repeat this evaluation to arrive at more quantitative results and comparisons.

14. Discussion

We already noted in the beginning that there are a number of related approaches to branch-and-bound algorithms using matchlists. Let us discuss these in more detail.

14.1. Hierarchical chamfer matching

Hierarchical chamfer matching (HCMA) [4] is an efficient and practical means for matching images under bounded error models. HCMA can be viewed as a kind of branch-and-bound algorithm that searches through the space of transformation parameters in a hierarchical manner. In HCMA, the exploration of transformation space is driven by sampling parameter space at different points. The condition that is imposed is that samples in parameter space should be close enough so that neighboring samples do not change the location of transformed model features by more than one pixel. This parameter step-length is approximated by computing partial derivatives of the location of the coordinates of the transformed model with respect to the transformation parameters. This is likely to be an excellent approximation in many practical problems; however, the techniques presented in [4] do not appear to guarantee a complete exploration of all possible solutions. By organizing the search about a recursive subdivision of transformation space, the algorithms presented in this paper can make such guarantees. The use of a pixel-based representation in HCMA also somewhat limits its applicability, since it may require the computation of large bitmaps for high resolution matching problems; matchlist-based methods are now routinely applied to matching geometric primitives and models in very large document images with subpixel accuracy.

14.2. Exact error propagation in correspondence methods

Correspondence-based methods, meaning methods like alignment [21] and interpretation trees [16], can be extended with exact error propagation methods like those described in [1]. Such methods will, given a set of correspondences between image and model features and given error bounds, predict reliable bounds on the possible locations of the remaining model features in the image. These locations can then be used to determine potentially matching image features. This set of potentially matching image features gives us an upper bound on the quality of match involving the chosen set of correspondences (traditional alignment gives us a lower bound). These correspond to the upper and lower bounds on the quality of match used by the branch-and-bound method described above and could be used in a correspondence-based branch-and-bound algorithm.

The key difference between such correspondence-based approaches and the transformation space-based approaches described here is that the transformation space-based approaches also give us a guaranteed and simple way of reducing the uncertainty. If we subdivide the region in transformation space, assuming sufficient smoothness of the imaging transform, the uncertainty in locations decreases predictably and steadily, and the subdivision into disjoint regions of transformation space also guarantees that each region is only explored once. Furthermore, subdividing transformation space to a chosen depth (corresponding to a chosen depth in the search tree) guarantees us a predictable (though not necessarily uniform) uncertainty in the resulting projection of model points into the image. As a practical consequence, transformation space-based branch-and-bound methods almost never return solutions that are not demonstrably optimal (optimality can be demonstrated for a solution by observing that its lower bound exceeds the upper bound of the next item on the priority queue).

In contrast, adding another correspondence to a set of correspondences may not reduce the uncertainty of the resulting alignment at all, or it may force the entire set of correspondences to become infeasible; as Grimson [16] has shown, the resulting search problem is hard in general. For correspondence-based searches, there is no guarantee that exploration to a certain depth will reduce uncertainty to a particular amount. The LiME dataset [3], used also in the benchmarks above, in fact, contains many test cases illustrating this point and appear to be particularly hard for correspondence-based approaches: because of the high degree of symmetry, many correspondences do not reduce uncertainty much at all.

It is possible that precise propagation of error bounds in a correspondence-based approach might also be used to construct algorithms with similar performance characteristics. For polygonal error bounds, [5] has already demonstrated such a correspondence-based algorithm, although it is far less efficient than transformation space based methods, and it has proven difficult to extend to other kinds of error bounds. This might be an interesting area for future research, since interpretation tree and correspondence-based methods have a number of desirable practical properties [16].

14.3. The RAST algorithm

The RAST (Recognition by Adaptive Subdivision of Transformation Space) algorithm [7,6] is a branch-and-bound algorithm relying on subdivisions and upper bounds in transformation space and is a precursor to matchlist-based methods described in this paper. In its original formulation [7,6], it applied to point matching under arbitrary convex polygonal error bounds and isoform (translation, rotation, scale) or affine planar transformations. Later, it was extended to matching under arbitrary transformations [9] and matching some parametric models [11]. Matchlists have their origins in the heuristic search methods extensively reviewed in the book by Grimson [16], as well as alignment methods [21]. Jurie [23] has extended the RAST algorithm to optimal matching under a Gaussian error model, also using a matchlist approach.

14.4. Hausdorff matching

Huttenlocher and Rucklidge [20,30] describe another approach to point matching under a quality of match function closely related to bounded error models. The authors consider a special transformation space, the space of translations and anisotropic scaling. Like HCMA, their approach relies on a discrete Voronoi surface and quickly rules out suboptimal transformations. The sampling in the method is guaranteed only to exclude suboptimal solutions; therefore, the solutions computed by their algorithm are globally optimal.

14.5. Point location and distance transform approaches

Both Hagedoorn and Veltkamp [18] and Mount et al. [26] have proposed branch-and-bound type algorithms for geometric matching using point location data structures for an efficient computation of $\hat{Q}(\mathbb{T})$. Hagedoorn and Veltkamp [18] also introduced the term “swept area,” which is a particularly lucid way of naming a key step in the computation of $\hat{Q}(T)$. Olson [29] proposes the use of point location data structures or distance transforms in these kinds of algorithms. These algorithms are true branch-and-bound algorithm, relying on a recursive subdivision of transformation space with explicit bounds, and guaranteeing the complete exploration of the parameter space. They are closely related to Borgefors's HCMA [4] and Huttenlocher and Rucklidge's [20] approach, if we view the Voronoi diagram as a simple and efficient point location data structure. But because, unlike the discrete Voronoi diagram, point location data structures can determine proximity without quantization, these algorithms can guarantee finding approximations to the optimal matches to any desired degree of accuracy.

While the use of point location data structures is appealing at first sight, as the experiments presented in this paper show, they do not usually result in speedups on common problems. This actually already becomes apparent when measuring execution profiles of matchlist-based code: the computation is usually not dominated by distance computations, and hence, the introduction of a point location data structure would not be expected to help. We have also seen the reason for this: without using matchlists, a branch-and-bound method needs to transform and examine every model point during each evaluation of $\hat{Q}(T)$; since transforming model points is a costly operation, this results in a substantial overhead relative to matchlist-based approaches even if performing point location took no time at all. However, this paper is the first that compares the two approaches empirically, and there may be interesting conditions or computational tricks to combine the approaches or overcome this issue, and this might be an interesting area for future research.

The use of point location data structures also significantly complicates the implementation of branch-and-bound algorithms for geometric matching, since it requires the careful implementation of an efficient geometric data structure, separately from the matching algorithm. When features involve orientation information, consists of line segments (as in Sections 9.4 and 13), or when error bounds are transform dependent or correlated, the resulting data structures require searches over three or higher dimensional spaces, possibly with variable metrics and topologies that differ from $\mathbb{R}^n$.

A matchlist-based approach, in contrast, effectively performs point location implicitly along with the matching algorithm and requires no separate data structure. An additional example of the utility of matchlist-based approaches can be found in [15], which describes two geometric algorithms that rely in an essential way on matchlists.

14.6. Using alignment to speed up branch-and-bound methods

Another idea that has been examined in the literature a number of times is the use of alignments to speed up matching in branch-and-bound algorithms. The idea is to use alignments to generate candidate transformations and then apply some form of local search to find local optima. If we choose the region over which we perform local search carefully, we can, in fact, guarantee that one of the local optima found in that way actually is also the global optimum [23]. Alternatively, we can use alignments to sample the space of potential solutions and be satisfied with finding a good or optimal solution with high probability [26]. If we incorporate the use of alignments to find starting points for further exploration into an algorithm that does not utilize matchlists, we would expect to see noticeable speedups. The reason is that the initial alignment cuts down the set of possible correspondences greatly, and methods that use this combination of alignment and local search generally take advantage of this fact.

However, once matchlists are incorporated into the branch-and-bound algorithms, it can be argued that using alignment to start local searches does not result in improved performance, and may actually slow down matching. One indication of this is the results presented in Section 12. From those results, we see that, since alignment and branch-and-bound methods scale in the same way (and apparently with similar constants), it seems likely that incorporating alignment to find starting points for the branch-and-bound method would result in better scaling behavior. Furthermore, if we assume that that the implementation of the alignment method and the implementation of branch-and-bound method described in Section 12 are of comparable quality, the timings described in that section suggest that performing branch-and-bound searching both globally and locally is actually less than even just picking and evaluating all the possible alignments.

We could explain an absence of improvements resulting from the incorporation of alignment methods into matchlist-based branch-and-bound methods as follows. A matchlist-based branch-and-bound algorithm will stop exploring regions of transformation space that do not contain good matches before it has needed to expend much computation on attempting precise geometric matches. Alignment-based methods, however, usually pick alignments without being able to take other information into account and only discover that particular alignments are poor or similar to each other after already having expended considerably effort on their evaluation.

In fact, historically, the development of matchlist-based branch-and-bound methods for geometric matching described in [6] resulted from attempts to quickly eliminate "uninteresting regions in transformation space" in correspondence-based algorithms with exact, optimal error propagation, described in [5]. The branch-and-bound method described here retains the matchlist from the correspondence-based algorithm but has become both simpler and more efficient by allowing transformations to be evaluated at points other than those determined by alignments.

15. Conclusions

While a number of different approaches have been proposed in the literature for branch-and-bound style geometric matching algorithms, using various data structures for estimating upper bounds, the experimental results presented in this paper strongly suggest that a matchlist approach is the simplest and most efficient approach under many common conditions. The experimental comparisons presented in this paper to commonly used approximations to geometric matching problems, like alignment and the Hough transform, also suggest that optimality really does yield matches that are of significantly higher quality than those commonly used approximations. Since it is also the simplest to implement among the branch-and-bound methods for geometric problems, it should probably be the first choice for anybody needing to solve geometric matching problems. In addition to the results presented here, the algorithm has been applied to many practical problems. The interested reader can find further information about applications to appearance-based

3D object recognition [8], character recognition [10], feature extraction [11], and document image analysis [14].

Of particular note are two novel applications of the matchlist-based approach to complex geometric optimization problems: the results on line segment matching described above and the two geometric algorithms described in [15].

The main goal of this paper has been to give readers enough detailed information to allow them to implement branch-and-bound methods to solve their own geometric matching problems easily and correctly. To this end, the paper contains some reference information about implementation strategies, transformation spaces, and quality of match measures. It also contains a number of experiments that highlight various behaviors of the algorithm and compare it empirically with alternative implementation strategies.

While this paper has examined a number of basic tradeoffs in the design of branch-and-bound algorithms for geometric matching, many properties of these algorithms still remain to be explored, both experimentally and theoretically.

Acknowledgments

Some of the results presented in this paper have been previously published as part of the author's thesis [7] and at several conference and workshops [6,9,13]. The author thanks Ross Beveridge for permission to use the line drawing database [3] and for his extensive comments on a first draft of the manuscript. The author also thanks the anonymous reviewers for their valuable comments on the manuscript.

References

[1] T.D. Alter, D.W. Jacobs, Uncertainty propagation in model-based recognition, Int. J. Comput. Vision (IJCV) 27 (2) (1998) 127–159.
[2] H.S. Baird, Model-Based Image Matching Using Location, MIT Press, Cambridge, MA, 1985.
[3] J.R. Beveridge, E.M. Riseman, How easy is matching 2D line models using local search? IEEE PAMI 19 (7) (1997).
[4] G. Borgefors, Hierarchical chamfer matching: a parametric edge matching algorithm, PAMI 10 (1988) 849–865.
[5] T.M. Breuel, An efficient correspondence based algorithm for 2D and 3D model based recognition, in: IEEE Conference on Computer Vision and Pattern Recognition, 1991.
[6] T.M. Breuel, Fast recognition using adaptive subdivisions of transformation space, in: IEEE Conference on Computer Vision and Pattern Recognition, 1992, pp. 445–451.
[7] T.M. Breuel, Geometric aspects of visual object recognition, Ph.D. Thesis, Massachusetts Institute of Technology, 1992. Also available as MIT AI Lab TR# 1374.
[8] T.M. Breuel, View-based recognition, in: MVA'92, IAPR Workshop on Machine Vision Applications, December, 1992, pp. 29–32.
[9] T.M. Breuel, Recognition by adaptive subdivision of transformation space (rast)—practical experiences and comparisons with the Hough transform, in: IEE Colloquium on Hough Transforms, Savoie Place, London, May, 1993.
[10] T.M. Breuel, Recognition of handprinted digits using optimal bounded error matching, in: ICDAR'93, 1993.

[11] T.M. Breuel, Finding lines under bounded error, Pattern Recognition 29 (1) (1996) 167–178.
[12] T.M. Breuel, Implicit manipulation of constraint sets for geometric matching under 2D translation and rotation,.in: Scandinavian Conference on Image Analysis (SCIA), Bergen, Norway, 2001.
[13] T.M. Breuel, A comparison of search strategies for geometric branch and bound algorithms, in: European Conference on Computer Vision, Copenhagen, Denmark, 2002.
[14] T.M. Breuel, Robust least square baseline finding using a branch and bound algorithm, in: Proceedings of the SPIE, The International Society for Optical Engineering, 2002.
[15] T.M. Breuel, Two algorithms for geometric layout analysis, in: Document Analysis Systems (DAS'02), Princeton, NJ, August, 2002.
[16] E. Grimson, Object Recognition by Computer, MIT Press, Cambridge, MA, 1990.
[17] M. Grötschel, L. Lovász, A. Schrijver, Geometric Algorithms and Combinatorial Optimization, Springer, Heidelberg, 1988.
[18] M. Hagedoorn, R.C. Veltkamp, Reliable and efficient pattern matching using an affine invariant metric, Int. J. Comput. Vision 31 (2/3) (1999) 203–225.
[19] D.P. Huttenlocher, G.A. Klanderman, W.J. Rucklidge, Comparing images using the Hausdorff distance, IEEE Trans. Pattern Anal. Machine Intell. 15 (9) (1993) 850–863.
[20] D.P. Huttenlocher, W.J. Rucklidge, A multi-resolution technique for comparing images using the Hausdorff distance, in: IEEE Conference on Computer Vision and Pattern Recognition, 1993, pp. 705–706.
[21] D.P. Huttenlocher, S. Ullman, Object recognition using alignment, in: International Conference on Computer Vision, London, England, June, IEEE, Washington, DC, 1987, pp. 102–111.
[22] W Wells III, Statistical approaches to feature-based object recognition, Int. J. Comput. Vision 21 (1/2) (1997) 63–98.
[23] F. Jurie, Solution of the simultaneous pose and correspondence problem using Gaussian error model, Comput. Vision Image Understanding 73 (3) (1999) 357–373.
[24] H. Kalviainen, E. Oja, L. Xu, Randomized Hough Transform (RHT): basic mechanisms algorithms and computational complexities, CVGIP: Image Understanding 57 (2) (1993) 131–154.
[25] D.M. Mount, S. Arya, ANN: A library for approximate nearest neighbor searching, in: CGC 2nd Annual Fall Workshop on Computational Geometry, 1997.
[26] D.M. Mount, N.S. Netanyahu, J. Le Moigne, Efficient algorithms for robust feature matching, Pattern Recognition 32 (1) (1999) 17–38.
[27] S. Nene, S. Nayar, H. Murase, Columbia object image library: Coil, Technical Report CUCS-006-96, Department of Computer Science, Columbia University, 1996.
[28] C.F. Olson, A probabilistic formulation for Hausdorff matching, in: IEEE Conference on Computer Vision and Pattern Recognition, 1998, pp. 150–156.
[29] C.F. Olson, Locating geometric primitives by pruning the parameter space, Pattern Recognition 34 (6) (2001) 1247–1256.
[30] W.J. Rucklidge, Efficiently locating objects using the Hausdorff distance, Int. J. Comput. Vision 24 (3) (1997) 251–270.
[31] X. Yi, O.I. Camps, Line-based recognition using a multidimensional Hausdorff distance, IEEE PAMI 21 (9) (1999).